# 面试题精选 ES6和新兴技术

## ES6新增了哪些特性？你通常使用哪些特性？

这些问题一般都是面对面的口答题目。

https://mp.weixin.qq.com/s?src=3&timestamp=1495525019&ver=1&signature=5Ejvg8tOKv\*y-zarozxWOzXLxw\*eVY8Ot7vIUv9BmqDbu5MU3zYEQaaUxocn-\*4I6oKvRtsz\*OtPgklhflKMoJWUV7SiHSqUXHxE\*j68s1um3vLxem\*Thj8Ua-JfPLwIY6oJVTwIyUEuwig9oezq02inhmve6WLgzuh9mZVQORA=

变量的定义，增加了let、const关键字。箭头函数。增加了字符串的方法、数组的方法。模板字符串，``。扩展操作符（...）。解构。Promise。引入了import、export关键字定义模块。类的定义class关键字。

https://mp.weixin.qq.com/s?src=3&timestamp=1495525019&ver=1&signature=UjH5AXilQDbckV1N3Kjh9RCGhWCbGALOc4NP2W56\*Aq828cHha3ZEJdaCqlPB5-SC76UjsEGC9yU5TM82B89-RAw0Wg1Ld\*uDu1KMG3rFFNktKBgRK5fHrmiEl05duGvtRMvYCGTRlIvY\*I2a1VYd7SKEcz8cxROmHHwtwbkoEo=

## 请写出3个ES6数组新方法，并举例说明

https://mp.weixin.qq.com/s?src=3&timestamp=1495525509&ver=1&signature=4gRZPRBAat7v1kITkINwECTAb\*IHd97gZOPsl2kQfxF5Zgc0hWbJ8Brm8Lv10AALfUDXXgSITMKzoaB-Q1Deio\*ALM4HX88U1SuXFzwm9\*VdlzWknvyTsgaKMaGwVlnLNtnZtg4MJsV8Ra2mKr0-sw==

● from()方法，可以将类数组对象转为数组

如果ES5中，想将类数组对象转为数组，可以借助Array的实例方法slice即可：

function fun(){

**var arr = Array.prototype.slice.call(arguments,0);**

console.log(arr);

}

fun("白板","幺鸡","二条","三桶","四饼");

ES5中使用Array.from()

function fun(){

var arr = Array.from(arguments);

console.log(arr);

console.log(Array.isArray(arr));

}

fun("白板","幺鸡","二条","三桶","四饼");

● copyWithin() 复制数组的内部的项

● find()方法 、findIndex() 里面是回调函数，返回第一个使回调函数为true的项/项的下标

var arr = [4,54,56,52,345,345,1345,346,324,45,34,23423];

var result = arr.find((item)=>{

return item >= 100 && item % 5 == 0;

});

console.log(result); //345

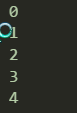
● entries()、keys()和values()

var arr = [345,234,23,65,45];

for(let key of arr.keys()){

console.log(key);

}



|  |
| --- |
| var arr = [345,234,23,65,45];  for(let [k,v] of arr.entries()){  console.log(k,v);  } |



● includes() 方法返回一个布尔值，表示某个数组是否包含给定的值

|  |
| --- |
| console.log([345,234,23,65,45].includes(23)); |

● forEach()、filter()、map()、reduce()函数

这四个函数数组数组的遍历，他们都是会遍历数组每一项。

forEach就是遍历数组中的每一项，和for循环类似，也如同\_.each();

|  |
| --- |
| [345,234,23,65,45].forEach((item) => {  console.log(item);  }); |

filter会返回一个新数组，新数组的项是原数组中return true的项

比如删除id为1的项，可以这样：

|  |
| --- |
| var arr = [{"id":0,"name":"小明"},{"id":1,"name":"小花"},{"id":2,"name":"小青"}];  arr = arr.filter((item) => {  return item.id != 1;  });  console.log(arr); |

map()会返回一个和原数组项数相同的项，每项都和原数组有映射关系：

比如返回一个新数组，新数组中的每一项都是原来项的平方：

|  |
| --- |
| var arr = [345,234,23,65,45].map((item) => {  return item \* item;  });  console.log(arr); |

reduce()的英语愿意是减少，本质是“迭代的遍历”。功能很强大：

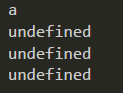
|  |
| --- |
| ["a","b","c","d","e"].reduce((a,b) => {  console.log("★");  }); |



因为少遍历最后一项，所以只有4个星星。

看看参数a：

|  |
| --- |
| ["a","b","c","d","e"].reduce((a,b) => {  console.log(a);  }); |



第一个参数表示遍历上一项的时候的return值，如果没有return就是undefined。

如果有return：

|  |
| --- |
| ["a","b","c","d","e"].reduce((a,b) => {  console.log(a);  return "哈哈";  }); |



b的值就是这一项的值，注意b不迭代第0项：

|  |
| --- |
| ["a","b","c","d","e"].reduce((a,b) => {  console.log(b);  }); |



## 完成一个’flatten’的函数，实现“拍平”一个多维数组为一维。示例如下：

var testArr1 = [[0, 1], [2, 3], [4, 5]];

var testArr2 = [0, [1, [2, [3, [4, [5]]]]]];

flatten(testArr1) // [0, 1, 2, 3, 4, 5]

flatten(testArr2) // [0, 1, 2, 3, 4, 5]

答案：

const flatten = arr => arr.reduce((pre, val) => pre.concat(Array.isArray(val) ? flatten(val) : val), []);

解析：

reduce特别活，先来几个热身：

求数组最大值：

|  |
| --- |
| var max;  [345,234,23,4545,65,45].reduce((a,b) => {  max = a > b ? a : b;  return max;  });  console.log(max); |

求和：

var sum;

[2,3,4,4,4].reduce((a,b) => {

sum = a + b;

return sum;

});

console.log(sum);

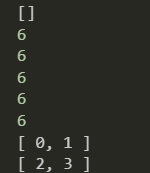
统计及格、不及格人数

|  |
| --- |
| var result = [45,365,45,6345,3,45,3,445].reduce((a,b)=>{  return b >= 60 ? {"jige" : a.jige + 1 , "bujige" : a.bujige} : {"jige" : a.jige , "bujige" : a.bujige + 1}  },{"jige" : 0 , "bujige" : 0})  console.log(result); |

拍平数组：

第一步：

|  |
| --- |
| var arr = [6,6,6,6,6,[0, 1], [2, 3], [4, 5]];  function flatten(arr){  arr.reduce((pre , value) => {  console.log(pre);  return value;  },[]); //[]表示默认值  }  flatten(arr); |



有迭代：

|  |
| --- |
| var arr = [[4,45,4],[0, 1], [2, 3], [4, 5]];  function flatten(arr){  **return arr.reduce((pre , value) => {**  **return pre.concat(Array.isArray(value) ? flatten(value) : value);**  **},[]);**  }  arr = flatten(arr);  console.log(arr); |

## 请谈谈ECMAScript 2015的Promise对象？

优雅的解决了回调黑洞的问题。

几篇公众号文章：

https://mp.weixin.qq.com/s?src=3&timestamp=1495525221&ver=1&signature=2S0hkhzGqw4wI35TZQsNdQvWBYvuaNQyT0e3mOdVN5lcBfcLGIale1VPrBty8RY0CQ910Ji\*23PabUqt3Uet\*ZLBVTSSEOabbM\*zL6wewtuGdKArqaKyqp-kU67qN8Q4WbY5TbsI2INv3OBF7IPYjhdYMPRnPbkOkvDF\*itMN0k=

https://mp.weixin.qq.com/s?src=3&timestamp=1495502961&ver=1&signature=YaSPg7qpu0Qva1\*0Fzsibkd\*dUmaJaaRaWFCm3-pr1PpkPfpryuJ7QaA9FtHpAOZDbBhNBo1o-kdbr2\*Aw1wbWk9LWDCJvpYywxLB10xsdOEFnnuUpQxZW7cmMtCPuwC8S5upNHcWaYQgcbdoo7Wev1yuMcbhB9cm9bp\*kXcPgw=

https://mp.weixin.qq.com/s?src=3&timestamp=1495502835&ver=1&signature=5Ejvg8tOKv\*y-zarozxWOzXLxw\*eVY8Ot7vIUv9BmqB4ovahgPeSN3jMiXgIhxA9dldtbRI8x89-m3ioo8e33SCeWE2d52ZCQI3EYImyIrn9l-DxPj1XZB3tx1BXpT3KV4mDWaUM1v6EVUyUGMu-Vg==

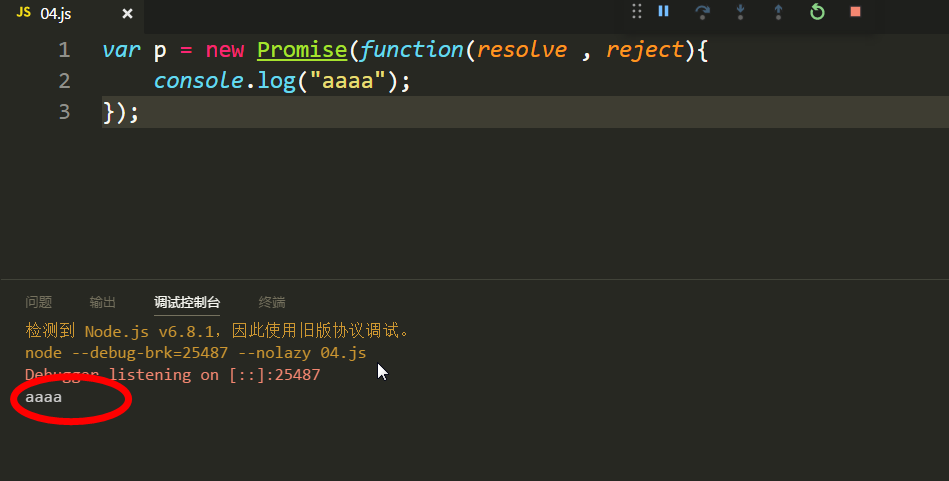
在Promise之前，在js中的异步编程都是采用回调函数和事件的方式，但是这种编程方式在处理复杂业务的情况下，很容易出现callback hell(回调地狱)，使得代码很难被理解和维护。

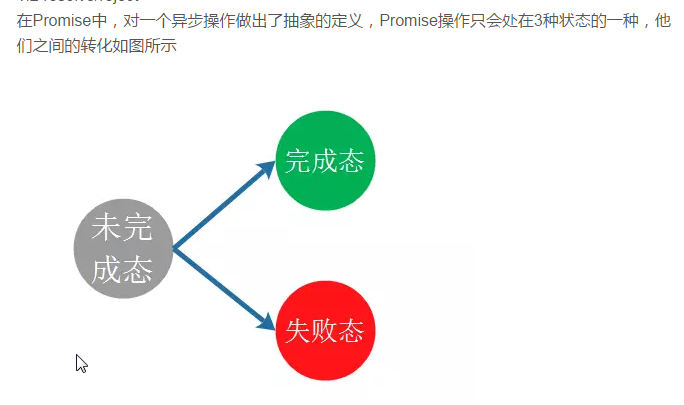
Promise就是改善这种情形的异步编程的解决方案，它由社区最早提出和实现，es6将其写进了语言标准，统一了用法，并且提供了一个原生的对象Promise。

基本的使用：



实例化Promise对象的时候，就会执行函数：





注意，这种状态的改变只会出现从未完成态向完成态或失败态转化，不能逆反。完成态和失败态不能互相转化，而且，状态一旦转化，将不能更改。

只有异步操作的结果可以决定当前是哪一种状态，任何其他操作都无法改变这个状态。这也是Promise这个名字的由来，它的英语意思是承诺，表示其他手段无法改变。

resolve就对应着完成态之后的操作

reject对应着失败态之后的操作

那么问题来了，then()方法有什么作用？resolve和reject又是从哪里传递过来的？

其实这两个问题是一个问题，在实例化一个Promise对象之后，我们调用该对象实例的then()方法传递的两个参数中:

第一个参数（函数）对应着完成态的操作，也就是resolve

第二个参数（函数）对应着失败态的操作，也就是reject

那就是说，在Promise中是通过then()方法来指定处理异步操作结果的方法。

实例，我们现在封装一个异步函数：

|  |
| --- |
| var fs = require("fs");  function loadFile(url){  return new Promise(function(resolve , reject){  fs.readFile(url , function(err,data){  if(err){  reject(err);  return;  }  resolve(data);  });  });  }  loadFile("./test.txt").then(function(data){  console.log(data.toString());  },function(err){  console.log(err);  }); |

resolve()函数只能接受一个实参，需要传回多个数据用对象：

|  |
| --- |
| var fs = require("fs");  var p1 = new Promise(function(resolve,reject){  fs.readFile("./test.txt",function(err,data){  if(err){  reject(err);  return;  }  resolve(**{**  **txt : data.toString(),**  **number : 22222**  **}**);  });  });    p1.then(function(**ret**){  console.log(**ret**.txt);  console.log(**ret**.number);  }); |

下面的例子中resolve()可以传入另一个Promise的实例：

test.txt中文字是“你好”

test2.txt中文字是“再见”

|  |
| --- |
| var fs = require("fs");  var p1 = new Promise(function(resolve,reject){  fs.readFile("./test.txt",function(err,data){  if(err){  reject(err);  return;  }  resolve({  txt : data.toString()  });  });  });  var p2 = new Promise(function(resolve,reject){  fs.readFile("./test2.txt",function(err,data){  if(err){  reject(err);  return;  }  resolve({  txt : data.toString() ,  next : p1  });  });  });    p2.then(function(ret){  console.log(ret.txt)  return ret.next;  }).then(function(ret){  console.log(ret.txt)  }); |

显示“你好”，就是p1中的文字。

优点：

可以看到Promise其实就是做了一件事情，**那就是对异步操作进行了封装，然后可以将异步操作以同步的流程表达出来，避免了层层嵌套的回调函数，同时提供统一的接口，使得控制异步操作更加容易**。

但是，Promise也有一些缺点：

**无法取消Promise，一旦新建它就会立即执行，无法中途取消。**

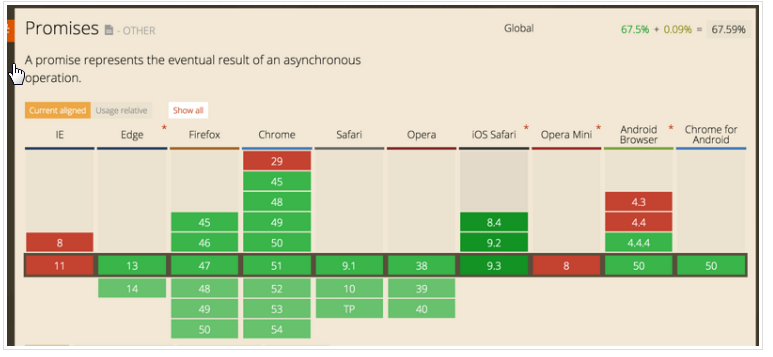
**如果不设置回调函数，Promise内部的错误不会反应到外部。**

**当处于未完成态时，无法得知目前进展到哪一个阶段。**

|  |
| --- |
| let pro1 = new Promise(function(resolve){  setTimeout(function () {  resolve('实例1操作成功');  },5000);  });    //创建实例pro2  let pro2 = new Promise(function(resolve){  setTimeout(function () {  resolve('实例2操作成功');  },1000);  });    Promise.all([pro1,pro2]).then(function(result){  console.log(result);  }); |

对于还不了解Async/await特性的同学，下面一段是一个“速成”培训。Async/await 是Javascript编写异步程序的新方法。以往的异步方法无外乎回调函数和Promise。但是Async/await建立于Promise之上。对于Javascript处理异步，是个老生常谈却历久弥新的话题：

在主体函数之前使用了async关键字。在函数体内，使用了await关键字。当然await关键字只能出现在用async声明的函数体内。该函数会隐式地返回一个Promise对象，函数体内的return值，将会作为这个Promise对象resolve时的参数。可以使用then方法添加回调函数。当函数执行的时候，一旦遇到await就会先返回，等到异步操作完成，再接着执行函数体内后面的语句。



## 请问webpack和gulp、grunt是什么？

https://mp.weixin.qq.com/s?src=3&timestamp=1495525367&ver=1&signature=dhxO6VpZ3sTTpOpX6\*0YxU9wHh56KdkzpAv-hP1IPbB\*a-ZAxteY8WbhV4Yjqmci9UliMddwgAOog4C9mQK8pQVcb1uTYWwq-VOTxivQJiZLaOV7WLvHrXOWfdsazaxU8ouq-pw7p4LW4yOydwD8Fb05AqNmf7T8wU-LY9s24rA=

## 如何深度克隆一个对象？

https://mp.weixin.qq.com/s?src=3&timestamp=1495525442&ver=1&signature=5Ejvg8tOKv\*y-zarozxWOzXLxw\*eVY8Ot7vIUv9BmqDzLPrai3Xr3jd5aRTcjXBSAz2VLt3ljVgkapebnbRDRFFFBd4gT3bm6oL\*2s3zfTjzXua3kOdkN9d\*JTnTBG1\*J-Thlvi3h7ivIVE9AIEitw==

## 请简述函数科里化

https://mp.weixin.qq.com/s?src=3&timestamp=1495525457&ver=1&signature=YaSPg7qpu0Qva1\*0Fzsibkd\*dUmaJaaRaWFCm3-pr1MLoWebAnIMfRPmB98E9a8rRN2TD3mnetxqFvK-gVu-FoU4mefthGwLsDYwseNBWtOx6-XW88FxeO4VsHYved-LG8iHdv\*bHwOdpcp8aRYgAsagqfcb-4X9zMd0KUpiwQE=

## Babel是什么，请介绍

## AMD和CMD规范